Unit testing tools for JavaScript & Python

Unit testing is a software testing approach where individual units or components of a program are tested independently to ensure they function correctly. It involves testing each unit in isolation to validate its behavior, often done through automated scripts, to detect bugs early in the development process.

To implement unit testing in our **E-Commerce System** project, we use **Jest** as the testing framework along with React Testing Library for frontend testing and for backend testing, we leverage **Pytest**.

Frontend Testing with JEST (JavaScript)

Jest Overview:

Jest is a delightful JavaScript Testing Framework with a focus on simplicity. It works seamlessly with projects using Babel, TypeScript, Node, React, and more. Jest provides a simple and efficient way to write tests, with features like built-in assertions, mocking capabilities, snapshot testing, and parallel test execution.

React Testing Library:

Alongside Jest, we utilize React Testing Library, a testing utility for React that encourages testing components in a way that resembles how they are used by the end-users.

Installation:

Jest and React Testing Library have been integrated into our project using the following installation:

```
npm install --save-dev jest @testing-library/react @testing-library/jest-dom
```

Let's get started by writing a test for a hypothetical function that adds two numbers.

1. **create a SumComponent.js file:**

```
import React from 'react';

export function sum(a, b) {
  return a + b;
}

function SumComponent(props) {
  const { a, b } = props;
  const result = sum(a, b);

  return <div>The sum of {a} and {b} is: {result}</div>;
}

export default SumComponent;
```

2. **Create a file named SumComponent.test.js. This will contain our actual test:**

```
import React from 'react';
import { render } from '@testing-library/react';
import SumComponent, { sum } from './SumComponent';

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});

test('renders sum component correctly', () => {
  const { getByText } = render(<SumComponent a={1} b={2} />);
  const resultElement = getByText('The sum of 1 and 2 is: 3');
  expect(resultElement).toBeInTheDocument();
});
```

3. **Create a jest.config.js file in your project's root directory:**

```
module.exports = {
    moduleNameMapper: {
        testEnvironment: 'jsdom',
        setupFilesAfterEnv: ['./jest.setup.js'],
    },
  };
```

4. **Add the following section to your package.json:**

```
{
    "scripts": {
      "test": "jest"
    }
  }
```

5. **Run the tests:**

Run the command **npm test** in your terminal, and Jest will execute the tests.

```
PASS    ./SumComponent.test.js
✓ adds 1 + 2 to equal 3 (5ms)
✓ renders sum component correctly (10ms)
```

This setup ensures a clean and organized structure for Jest and React Testing Library integration.

Backend Testing With Pytest (Python)

Pytest:

a powerful testing framework for Python that seamlessly integrates with Django, ensuring comprehensive unit testing coverage across both frontend and backend components.

Installation:

To integrate pytest into our Django project, the installation command is:

pip install pytest

We can write tests for Django views, models, or any other components using pytest. For instance, a test for a Django view might look like:

```
# test_views.py
from django.test import TestCase

class MyViewTestCase(TestCase):
    def test_view_returns_correct_status_code(self):
        response = self.client.get('/my-url/')
        self.assertEqual(response.status_code, 200)
```

By combining Jest and React Testing Library for frontend testing and pytest for Django backend testing, our project maintains a comprehensive unit testing strategy, ensuring the reliability and functionality of both frontend and backend components.